Housing Resources

Level: Census Tract Level

Definition: This dataset includes census tract-level data concerning housing in Metropolitan Detroit. The data includes: 1) Total housing units and total mortgages in the tract; 2) Land use; 3) Real estate information (foreclosures, sales transactions, and home values); 4) Vacant housing; 5) Housing age and available facilities; 6) Housing condition; and 7) Spatial measures of subsidized housing in the tract.

Dataset Version: 1.1

Dataset Release History: First Published in 2017

Initial Release: 11/01/2017

Dataset Name: NEIGHBORHOOD_EFFECTS_HOUSING _CENSUS_TRACT_LEVEL

Sources: American Community Survey, Variables derived from RealtyTrac raw data, Data Driven Detroit, United States Postal Service, United States Department of Housing & Urban Development (HUD)

Other Notes:

- Spatial coverage for foreclosure, home value and transaction data: 9 counties in Metropolitan Detroit (Genesee, Lapeer, Livingston, Oakland, Macomb, Monroe, Washtenaw, Wayne, and St. Clair)
- Spatial coverage for land use and housing condition: City of Detroit only.
- Spatial coverage for vacant housing, housing age and available facilities, and subsidized housing data: 10-county Detroit-Warren-Ann Arbor Combined Statistical Area.
- Python code to create total housing units and mortgages and real estate-related variables is available as an Appendix at the end of this codebook.

Variables	Year	Description	Notes
Identifier			
GEOID10		Primary key	Census Tract in 2010
CountyCode		County Code	
CountyName		County name	
DataYear		Data Year to which the row refers	

1) Total housing units and total mortgages			
All_Housing	2006 to 2010 and 2007 to 2011	An estimate of total housing units from American Community Survey (ACS) dataset	See appendix of Python code to create variables. line 791-842 for details
Real_Estate_Owned_Mortgages	2006 to 2010 and 2007 to 2011	An estimate of housing units with a mortgage from American Community Survey (ACS) dataset	See appendix of Python code to create variables. line 791-842
Real_Estate_Owned_Mortgages _By_All_Housing	2006 to 2010 and 2007 to 2011	Proportion of housing with mortages REAL_ESTATE_OWNED divided by ALL_HOUSING	See appendix of Python code to create variables. line 1093-1099
2) Land Use			
Parcel_Count	2013-2014	Number of land parcels in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
Total_Surveyed	2013-2014	Number of land parcels surveyed in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
StructureYes	2013-2014	Number of land parcels surveyed as having a structure in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
Use_Commercial	2013-2014	Number of land parcels with structures surveyed as commercial in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
PctStructure_Commercial	2013-2014	Number of land parcels with structures surveyed as commercial in census tract divided by the number of parcels with structures	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
Use_Industrial	2013-2014	Number of land parcels with structures surveyed as industrial in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
PctStructure_Industrial	2013-2014	Number of land parcels with structures surveyed as industrial in census tract divided by the number of parcels with structures	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.

Use_Residential	2013-2014	Number of land parcels with structures surveyed as residential in census tract	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
PctStructure_Residential	2013-2014	Number of land parcels with structures surveyed as residential in census tract divided by the number of parcels with structures	Source: Data Driven Detroit, 2014 Motor City Mapping Project. Spatial Extent: City of Detroit only.
3) **Real Estate Information**			
Foreclosures	2006 to 2010 and 2007 to 2011	Fixed number of Foreclosures that have a valid address in the census tract	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 693-765
Foreclosures_ By_Real_Estate_Owned_Mortgages	2006 to 2010 and 2007 to 2011	Foreclosure rate as a proportion of mortgages in census tract FORECLOSURES divided by REAL_ESTATE_OWNED_MORTGAGES	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 791-842
Foreclosures_ By_All_Housing	2006 to 2010 and 2007 to 2011	Foreclosure rate as a proportion of mortgages in census tract FORECLOSURES divided by ALL_HOUSING	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 791-842
Sales_Transactions	2006 to 2010 and 2007 to 2011	A 5-year estimate of land parcel-based sales transactions in the census tract	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 999-1042
Sales_Transactions_By_All_Housing	2006 to 2010 and 2007 to 2011	Real estate transaction rate by census tract SALES_TRANSACTIONS divided by ALL_HOUSING	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 999-1042
Val_Ass	2006 to 2010 and 2007 to 2011	An average value from all property values in the census tract, as assessed by Tax Assessor	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 861-930
Val_Market	2006 to 2010 And	An average market value from all property values in the census tract, as assessed by Tax Assessor	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at:

	2007 to 2011		line 861-930
Val_Ass_15	2006 to 2010 and 2007 to 2011	Average tax assessed home values in census tract, data converted to 2015 US dollars VAL_ASS in 2015 U.S. dollars	$(Val\_ASS)_{2010} *$ $(Inflation\ rate\ from\ 2010\ to\ 2015)$ Inflation rate from 2010 to 2015: 12.04% Inflation rate from 2011 to 2015: 8.61%. Inflation rate from 2012 to 2015: 6.41%. Inflation rate from 2013 to 2015: 4.87%. Inflation rate from 2014 to 2015: 1.57%. Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 861-930
Val_Market_15	2006 to 2010 and 2007 to 2011	Average market value of homes in census tract as assessed by, data converted to 2015 US dollars VAL_MARKET in 2015 U.S. dollars	Variables derived from parcel-level data from RealtyTrac. See Appendix for Python code used to create variables. See code at: line 861-930
4) Vacant Housing			
Ams_Res	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of Addresses - Residential	Source: United States Postal Service.
Ams_Bus	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of Addresses - Business	Source: United States Postal Service.
Res_Vac	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of Vacant Addresses – Residential	Source: United States Postal Service. These are addresses that letter carriers on urban routes have identified as not collecting their mail for 90 days or longer.
Bus_Vac	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of Vacant Addresses - Business	Source: United States Postal Service. These are addresses that letter carriers on urban routes have identified as not collecting their mail for 90 days or longer.
Nostat_Res	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of No-Stat Addresses - Residential	Source: United States Postal Service. No-Stat is defined as an address that has one of several characteristics. It is most frequently associated with new-construction properties that have not yet

			been occupied or blighted properties that have been identified as not likely to be occupied for some time, or Rural Route addresses vacant for 90 days or longer.
Nostat_Bus	2010 (Quarter 1) and 2014 (Quarter 3)	Total Count of No-Stat Addresses - Business	Source: United States Postal Service. No-Stat is defined as an address that has one of several characteristics. It is most frequently associated with new-construction properties that have not yet been occupied or blighted properties that have been identified as not likely to be occupied for some time, or Rural Route addresses vacant for 90 days or longer.
5) Housing age and available facilities			
Age_HD01_VD01	2009 to 2013	MEDIAN YEAR STRUCTURE BUILT: Estimate; Median year structure built	Source: American Community Survey.
Age_HD02_VD01	2009 to 2013	MEDIAN YEAR STRUCTURE BUILT: Margin of Error; Median year structure built	Source: American Community Survey.
Age_HD03_VD01	2009 to 2013	MEDIAN YEAR STRUCTURE BUILT: Standard Error; Median year structure built	Source: American Community Survey.
Kitchen_HD01_VD01	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Estimate; Total:	Source: American Community Survey.
Kitchen_HD02_VD01	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total:	Source: American Community Survey.
Kitchen_HD03_VD01	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Standard Error; Total:	Source: American Community Survey.
Kitchen_HD01_VD02	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Estimate; Total: - Complete kitchen facilities	Source: American Community Survey.
Kitchen_HD02_VD02	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total: - Complete kitchen facilities	Source: American Community Survey.
Kitchen_HD03_VD02	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Standard Error; Total: - Complete kitchen facilities	Source: American Community Survey.
Kitchen_HD01_VD03	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Estimate; Total: - Lacking complete kitchen facilities	Source: American Community Survey.
Kitchen_HD02_VD03	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total: - Lacking complete kitchen facilities	Source: American Community Survey.
Kitchen_HD03_VD03	2009 to 2013	KITCHEN FACILITIES FOR ALL HOUSING UNITS: Standard Error; Total: - Lacking complete kitchen facilities	Source: American Community Survey.
Plumbing_HD01_VD01	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Estimate; Total:	Source: American Community Survey.
Plumbing_HD02_VD01	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total:	Source: American Community Survey.

Plumbing_HD01_VD02	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Standard Error; Total:	Source: American Community Survey.
Plumbing_HD02_VD02	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Estimate; Total: - Complete plumbing facilities	Source: American Community Survey.
Plumbing_HD03_VD02	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total: - Complete plumbing facilities	Source: American Community Survey.
Plumbing_HD01_VD03	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Standard Error; Total: - Complete plumbing facilities	Source: American Community Survey.
Plumbing_HD02_VD03	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Estimate; Total: - Lacking complete plumbing facilities	Source: American Community Survey.
Plumbing_HD03_VD03	2009 to 2013	PLUMBING FACILITIES FOR ALL HOUSING UNITS: Margin of Error; Total: - Lacking complete plumbing facilities	Source: American Community Survey.
6) Housing condition			
DRPS_2009_AvgCond	2009	2009 average housing condition by census tract	Source: Data Driven Detroit, 2014 Motor City Mapping and Data Driven Detroit, Detroit Residential Parcel Survey, 2009. Spatial coverage: City of Detroit only.
MCM_2014_AvgCond	2014	2014 average housing condition by census tract	Source: Data Driven Detroit, 2014 Motor City Mapping and Data Driven Detroit, Detroit Residential Parcel Survey, 2009. Spatial coverage: City of Detroit only.
7) Subsidized Housing			
SubHou_Density	2015	Kernel Density of Subsidized Housing Locations by census tract	Computed based on data from the National Housing Preservation Database. Calculated using ArcMap software's Kernel Density Tool using Zonal Statistics. Radius setting =1320 feet. Spatial Extent: 10-county Detroit-Warren-Ann Arbor Combined Statistical Area.
SubHou_Distance	2015	Average Distance from Subsidized Housing Locations within census tract	Computed based on data from the National Housing Preservation Database. Calculated using ArcMap software's Kernel Density Tool using Zonal Statistics. Radius setting =1320 feet.

			Spatial Extent: 10-county Detroit-Warren-Ann Arbor Combined Statistical Area.

Appendix – python code used to create foreclosure, transaction and home value data as derived from RealtyTrac raw data

```
import re
import itertools
import datetime
import sqlite3 as sqlite
import requests
import urllib2
import json
import os
import traceback
import pandas as pd
import numpy as np
from inflation_calc.inflation import Inflation

#This is for txt -> multiple csv files
#and multiple csv files -> one big files
def columnInformation(csvFileFullName,k):
	#Getting each column's information from csvFileFullName
	#k==0 for Foreclosure
	#k==1 for Recorder
	#k==2 for TaxAssessor
	with open(csvFileFullName,'r') as fl:
		if k==0:
			#fl.readlines() or .read() gives one big string, so we are going to seperate by '\n'
			of=fl.read().splitlines()[2:69] #first 2 lines are unnecessary and we only need upto line 69.
		elif k==1:
			of1=fl.read().splitlines()[2:47] #first 2 lines are unnecessary and we only need upto line 47.
			with open(csvFileFullName,'r') as fl:
```

```
                                of2=fl.read().splitlines()[64:108]
                        of=of1+of2
                else:
                        of=fl.read().splitlines()[2:185] #first 2 lines are unnecessary and we only need upto line
        z=[re.search(r'[0-9].*[0-9],',i).group().split(',')[:-1] for i in of]
        z=zip(*z)
        #z[0] is column number
        #z[1] is headers
        #z[-1] is amount space that each column has.
        return z

def makingSingleFileForEach():
        recorderFileList=['UMichRecorder1a.csv','UMichRecorder1b.csv','UMichRecorder1c.csv','UMichRecorder2a.csv','UMichRecor
der2b.csv','UMichRecorder2c.csv','UMichRecorder3a.csv','UMichRecorder3b.csv','UMichRecorder3c.csv']
        taxAsseossorFileList=['UMichTaxAssessor1a.csv','UMichTaxAssessor1b.csv','UMichTaxAssessor1c.csv','UMichTaxAssessor2a.c
sv','UMichTaxAssessor2b.csv','UMichTaxAssessor2c.csv']
        keysList=['1a','1b','1c','2a','2b','2c','3a','3b','3c']
        with open('UMichTaxAssessor_Total.csv','w') as wo:
                with open('UMichTaxAssessor1a.csv','r') as rc:
                        for i in rc.readlines():
                                wo.write(i)
                for i in range(5):
                        with open(taxAsseossorFileList[i+1],'r') as rc:
                                for j in rc.readlines()[1:]:
                                        wo.write(j)
        with open('UmichRecorder_Total.csv','w') as wo:
                with open('UMichRecorder1a.csv','r') as rc:
                        for i in rc.readlines():
                                wo.write(i)
                for i in range(8):
                        with open(recorderFileList[i+1],'r') as rc:
```

```
                                        for j in rc.readlines()[1:]:
                                                wo.write(j)
        return 'Done'

def makingTxtfile(rrr,www,hhh,c):
        #rrr = CSV full file name
        #www = name textfile
        #hhh = headers
        #c==0 for Forclosure
        #c==1 for Recorder
        #c==2 for TaxAssessor

        #Getting each column's information from csvFileFullName
        with open(rrr,'r') as fl:
                if c==0:
                                        #fl.readlines() or .read() gives one big string, so we are going to seperate by '\n'
                                        of=fl.read().splitlines()[2:69] #first 2 lines are unnecessary and we only need upto line 69.
                elif c==1:
                        of1=fl.read().splitlines()[2:64] #first 2 lines are unnecessary and we only need upto line 47.
                        with open(rrr,'r') as fl:
                                of2=fl.read().splitlines()[64:108]
                        s=''
                        for i in of1[45:]:
                                s+=i
                        of1[44]=of1[44]+s
                        of=of1[:45]+of2
                else:
                        of=fl.read().splitlines()[2:185]
        with open('UMich'+www+'.txt','w') as wo:
                ck=0
                for i in of:
```

```
93                              if re.search(r'(Yes|No)(.*)',i):
94                                      k = re.search(r'(Yes|No)(.*)',i).group(2)[1:]
95                                      if re.search(r'\s{2,}',k):
96                                              gone = re.search(r'\s{2,}',k).group()
97                                              k = k.replace(gone,'')
98                              else:
99                                      k = ''
100                             wo.write("\n".join([hhh[ck],'\t'+k])+'\n\n\n')
101                             ck+=1
102             return 'DONE'
103
104     def checkingRecords(k,col,dd):
105             #imput: k = each file's unique name, col=column number, dd=dictionary
106             with open('UMich'+k+'.csv') as dk:
107                     for i in dk.readlines()[1:]:
108                             if i.split(',')[col].lower() in dd:
109                                     dd[i.split(',')[col].lower()]+=1
110                             else:
111                                     dd[i.split(',')[col].lower()]=1
112             return dd
113
114     def forRecorderOrTaxAssessorCSV(rrr,www,columnInformation):
115             eachColumnSpace=[int(i) for i in columnInformation[-1]]
116             #rrr is file to read, www is a list of file names (3), eachcolumnspace is each column information
117             with open(rrr,'r') as rc:
118                     totalNumberOfRows=len(rc.readlines())
119             remainder=totalNumberOfRows%3
120             totalNum=totalNumberOfRows-remainder
121             oneChunk=totalNum/3
122             splitFiles=[oneChunk,2*oneChunk]#we only need to check 2 points to split them into 3 parts
123
```

```
124         with open(rrr,'r') as rc:
125             with open('UMich'+www[0]+'.csv','w') as wo:
126                 wo.write(','.join(columnInformation[1])+'\n')
127                 for i in rc.readlines()[0:splitFiles[0]]:
128                     eachList=[]
129                     for j in eachColumnSpace:
130                         eachList.append(i[:j])
131                         i = i[j:]
132                     #Last element of eachList is '\r\n'
133                     eachList=eachList[:-1]
134                     #We want to grab only data only or empty variable
135                     eachList=[re.search(r'([^\s].*[^\s]|[^\s])|[ ]+', i).group() for i in eachList]
136                     #For empty variables, we are replacing spaces with '' none.
137                     for i in range(len(eachList)):
138                         if re.search(r'^\s+', eachList[i]):
139                             eachList[i]=eachList[i].replace(' ','')
140                     #Replacing commas with '' none.
141                     eachList=[i.replace(',','') for i in eachList]
142                     wo.write(",".join(eachList)+'\n')
143 
144         with open(rrr,'r') as rc:
145             with open('UMich'+www[1]+'.csv','w') as wo:
146                 wo.write(','.join(columnInformation[1])+'\n')
147                 for i in rc.readlines()[splitFiles[0]:splitFiles[1]]:
148                     eachList=[]
149                     for j in eachColumnSpace:
150                         eachList.append(i[:j])
151                         i = i[j:]
152                     #Last element of eachList is '\r\n'
153                     eachList=eachList[:-1]
154                     #We want to grab only data only or empty variable
```

```
                                        eachList=[re.search(r'([^\s].*[^\s]|[^\s])|[ ]+', i).group() for i in eachList]
                                        #For empty variables, we are replacing spaces with '' none.
                                        for i in range(len(eachList)):
                                                if re.search(r'^\s+', eachList[i]):
                                                        eachList[i]=eachList[i].replace(' ','')
                                        #Replacing commas with '' none.
                                        eachList=[i.replace(',','') for i in eachList]
                                        wo.write(",".join(eachList)+'\n')

        with open(rrr,'r') as rc:
                with open('UMich'+www[2]+'.csv','w') as wo:
                        wo.write(','.join(columnInformation[1])+'\n')
                        for i in rc.readlines()[splitFiles[1]:]:
                                eachList=[]
                                for j in eachColumnSpace:
                                        eachList.append(i[:j])
                                        i = i[j:]
                                #Last element of eachList is '\r\n'
                                eachList=eachList[:-1]
                                #We want to grab only data only or empty variable
                                eachList=[re.search(r'([^\s].*[^\s]|[^\s])|[ ]+', i).group() for i in eachList]
                                #For empty variables, we are replacing spaces with '' none.
                                for i in range(len(eachList)):
                                        if re.search(r'^\s+', eachList[i]):
                                                eachList[i]=eachList[i].replace(' ','')
                                #Replacing commas with '' none.
                                eachList=[i.replace(',','') for i in eachList]
                                wo.write(",".join(eachList)+'\n')
        return "Done"

def onlyForeclosureCSV(rrr,www,columnInformation):
```

```
        eachColumnSpace=[int(i) for i in columnInformation[-1]]
        #rrr is file to read, www is a file that we want to write, eachColumnSpace is each column information
        with open(rrr,'r') as fc:
                with open(www,'w') as wo:
                        wo.write(','.join(columnInformation[1])+'\n')
                        for i in fc.readlines():
                                eachList=[]
                                for j in eachColumnSpace:
                                        eachList.append(i[:j])
                                        i = i[j:]
                                #Last element of eachList is '\r\n'
                                eachList=eachList[:-1]
                                #We want to grab only data only or empty variable
                                eachList=[re.search(r'([^\s].*[^\s]|[^\s])|[ ]+', i).group() for i in eachList]
                                #For empty variables, we are replacing spaces with '' none.
                                for i in range(len(eachList)):
                                        if re.search(r'^\s+', eachList[i]):
                                                eachList[i]=eachList[i].replace(' ','')
                                #Replacing commas with '' none.
                                eachList=[i.replace(',','') for i in eachList]
                                wo.write(",".join(eachList)+'\n')
        return "Done"

#This is for sqlite.
def makingListForACertainColumn_Integer(fullName,listOfIndexNumb):
        #fullName:file full name
        #listOfIndexNumb: list of index number
        #returns a single list contains len(listOfIndexNumb) number of tuples
        with open(fullName,'r') as fl:
                ff=fl.readlines()
        return [tuple(map(lambda j: int(i.split(',')[j]) if len(i.split(',')[j])!=0 else 0, listOfIndexNumb)) for i in ff[1:]]
```

```
#This is for sqlite.
def makingListForACertainColumn_Text(fullName,listOfIndexNumb):
	#fullName:file full name
	#listOfIndexNumb: list of index number
	#returns a single list contains len(listOfIndexNumb) number of tuples
	with open(fullName,'r') as fl:
		ff=fl.readlines()
	return [tuple(map(lambda j: i.split(',')[j], listOfIndexNumb)) for i in ff[1:]]

def gettingCensusTract(directory_k,k,starting=0,anyAdditionalName=''):
	#directory_k is a string ex) 'Mar/subfiles'
	#k is a list, contain how many files you want to request data
	#starting is an integer (a starting index number)
	#anyAdditionalName is a string for file name
	print os.listdir(directory_k)[k[0]:k[1]]
	for filename in os.listdir(directory_k)[k[0]:k[1]]:
		print filename
		print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
		dict_id_coordinates={}
		dict_id_row={}
		rowNumber=1
		dict_id_propertyValues={}
		with open(directory_k+filename,'r') as ck:
			ck=ck.readlines()
		for i in ck[1:]:
			z=i.replace('\r\n','').replace('\n','').split(',')
			dict_id_row[z[1]]=[rowNumber]
			rowNumber+=1
			dict_id_propertyValues[z[1]]=(z[6],z[7])
			dict_id_coordinates[z[1]]=(z[-1],z[-2])
		base='http://data.fcc.gov/api/block/2010/find'
```

```
248                     ak= open('Zip_Year_Tract_'+anyName+filename,'w')
249                     ak.write('Tract,Zip,Year,saPropertyId,Val_ass,Val_market\n')
250                     c=1
251                     #for i in sorted(dict_id_coordinates.keys()):
252                     for i in dict_id_coordinates:
253                             if c>starting:
254                                     zipp=ck[dict_id_row[i][0]].split(',')[4]#zip
255                                     yearr=ck[dict_id_row[i][0]].split(',')[5]#year
256                                     #Getting tract information for each housing unit
257                                     option={'latitude': float(dict_id_coordinates[i][0]),'longitude':float(dict_id_coordinates[i][1])}
258                                     tractt=re.search(r'<Block FIPS="([0-9]{11})',requests.get(url=base, params=option).text).group(1)
259                                     val1=dict_id_propertyValues[i][0]
260                                     val2=dict_id_propertyValues[i][1]
261                                     ak.write("{},{},{},{},{},{}\n".format(tractt,zipp,yearr,i,val1,val2))
262                                     print c,len(ck)
263                             c+=1
264                     ak.close()
265             return 'Done'
266
267     def mergingCensusTractFiles(k,fileName):
268             #k is an integer, either 0 or 1 due to having 'DS_...' file
269             #fileName is a string ex) 'homeValues.txt'
270             initialOne = pd.read_csv('Mar/pct/'+os.listdir('Mar/pct')[k])
271             #print initialOne.columns.to_series().groupby(initialOne.dtypes).groups
272             for i in os.listdir('Mar/pct')[k+1:]:
273                     each=pd.read_csv('Mar/pct/'+i)
274                     initialOne = initialOne.append(pd.DataFrame(data = each), ignore_index=True)
275             initialOne = initialOne[initialOne.Zip != 0.0]
276             colls=['Zip','Year','saPropertyId','Val_ass','Val_market']
277
278             initialOne[colls]=initialOne[colls].applymap(np.int64)
```

```
279             initialOne.to_csv('Mar/pct/'+fileName,index=False)
280             return 'Done'
281
282     def usingInflationAPI(fileName,whereToSave):
283             #fileName is a string ex) 'homeValues.txt'
284             #whereToSave is a string ex) 'Mar/NE_PropertyValues.txt'
285
286             # Create a new Inflation instance
287             inflation = Inflation()
288             infla={}
289             # How many US $ would I need in 2015 to pay for what cost $1 in eachYear
290             for i in range(11):
291                     eachYear=i+2004
292                     infla[eachYear]=inflation.inflate(1, datetime.date(2015,1,1), datetime.date(eachYear,1,1), 'United States')
293             print infla
294             dfp=pd.read_csv('Mar/pct/'+fileName)
295
296             dfp=dfp.sort(['Zip','Tract','Year'])
297             dfp=dfp.reset_index(drop=True)
298             dfp=dfp.rename(columns =
299     {'Tract':'GEOID10','Val_ass':'VAL_ASS','Val_market':'VAL_MARKET','saPropertyId':'PROPERTY_ID','Zip':'ZIP','Year':'YEAR'})
300
301             dfp['VAL_ASS_15']=0.0
302             dfp['VAL_MARKET_15']=0.0
303
304             for i in infla:
305                     print i
306                     dfp.ix[dfp['YEAR']==i,'VAL_ASS_15'] = dfp.ix[dfp['YEAR']==i,'VAL_ASS'] * infla[i]
307                     dfp.ix[dfp['YEAR']==i,'VAL_MARKET_15'] = dfp.ix[dfp['YEAR']==i,'VAL_MARKET'] * infla[i]
308
309             dfp = dfp[dfp['GEOID10'].astype(str).str.startswith('26')]
```

```
        dfp['GEOID10']=dfp['GEOID10'].astype(int)
        dfp.to_csv(whereToSave,index=False)
        return 'Done'

def replacingPdColumn(df,newname,oldname):
        df[newname] = df[oldname]
        del df[oldname]
        return 1


##########################################################################################################
#########################    Step 1    ###################################################################
##########################################################################################################

################    Creating UMichForeclosure_Total.csv, UMichRecorder_Total.csv, UMichTaxAssessor_Total.c
################    From original REALTYTRAC txt files

#######Foreclosure
#Getting each column's information from REALTYTRAC DLP 3.0 Foreclosure Layout.xlsx
eachColumn = columnInformation('REALTYTRAC DLP 3.0 Foreclosure Layout.csv',0)
ignoreMe = onlyForeclosureCSV('University_of_Michigan_Foreclosure_001.txt','UMichForeclosure_Total.csv',eachColumn)

#######Recorder
#Getting each column's information from REALTYTRAC DLP 3.0 Recorder Layout.xlsx
eachColumn = columnInformation('REALTYTRAC DLP 3.0 Recorder Layout.csv',1)
ignoreMe =
forRecorderOrTaxAssessorCSV('University_of_Michigan_Recorder_001.txt',['Recorder1a','Recorder1b','Recorder1c'],eachColumn)
ignoreMe =
forRecorderOrTaxAssessorCSV('University_of_Michigan_Recorder_002.txt',['Recorder2a','Recorder2b','Recorder2c'],eachColumn)
ignoreMe =
forRecorderOrTaxAssessorCSV('University_of_Michigan_Recorder_003.txt',['Recorder3a','Recorder3b','Recorder3c'],eachColumn)
```

```
341
342     #######TaxAssessor
343     #Getting each column's information from REALTYTRAC DLP 3.0 Assessor NO Geo Layout.xlsx
344     eachColumn = columnInformation('REALTYTRAC DLP 3.0 Assessor NO Geo Layout.csv',2)
345     ignoreMe =
346     forRecorderOrTaxAssessorCSV('University_of_Michigan_TaxAssessor_001.txt',['TaxAssessor1a','TaxAssessor1b','TaxAssessor1c'],eac
347     hColumn)
348     ignoreMe =
349     forRecorderOrTaxAssessorCSV('University_of_Michigan_TaxAssessor_002.txt',['TaxAssessor2a','TaxAssessor2b','TaxAssessor2c'],eac
350     hColumn)
351
352     #making UmichRecorder_Total +UmichTaxAssessor_Total file.
353     ignoreMe = makingSingleFileForEach()
354
355     #################     Checking if data has right number of records
356
357     #######Foreclosure
358     kk = checkingRecords('Foreclosure',col=6,dd={})
359     print sum(kk.values())
360     print sorted(kk.items(), key= lambda x: x[0])
361     #Total: 429875
362     #[('genesee', 31185)
363     #('lapeer', 4885),
364     #('livingston', 7556),
365     # ('macomb', 60447),
366     # ('monroe', 7582),
367     # ('oakland', 73751),
368     # ('saint clair', 9886),
369     # ('washtenaw', 13098),
370     # ('wayne', 221197)]
371
```

```
#######Recorder
lr = ['Recorder'+j for j in ['1a','1b','1c','2a','2b','2c','3a','3b','3c']]
for i in lr:
	if i =='Recorder1a':
		subs = checkingRecords(i,col=7,dd={})
	else:
		final = checkingRecords(i,col=7,dd=subs)
		subs = final
print sum(final.values())
print sorted(final.items(), key= lambda x: x[0])
# Total: 7056997
# [('genesee', 427178),
# ('lapeer', 58920),
# ('livingston', 349521),
# ('macomb', 774258),
# ('monroe', 111704),
# ('oakland', 2186468),
# ('st. clair', 125686),
# ('washtenaw', 479496),
# ('wayne', 2543766)]

#######TaxAssessor
lta = ['TaxAssessor'+j for j in ['1a','1b','1c','2a','2b','2c']]
for i in lta:
	if i =='TaxAssessor1a':
		subs = checkingRecords(i,col=6,dd={})
	else:
		final = checkingRecords(i,col=6,dd=subs)
		subs = final
print sum(final.values())
print sorted(final.items(), key= lambda x: x[0])
```

```
# Total: 2290956
# [('genesee', 195890),
# ('lapeer', 44837),
# ('livingston', 90088),
# ('macomb', 330502),
# ('monroe', 75122),
# ('oakland', 485889),
# ('saint clair', 82512),
# ('washtenaw', 146519),
# ('wayne', 839597)]
```

```
#############################################################################################################
#########################    Step 2    ######################################################################
#############################################################################################################

################    Txt file for the column information(description)

#Getting each column's information from REALTYTRAC DLP 3.0 Foreclosure Layout.xlsx
eachColumn = columnInformation('REALTYTRAC DLP 3.0 Foreclosure Layout.csv',0)
headers = eachColumn[1]#headers
ignoreMe = makingTxtfile('REALTYTRAC DLP 3.0 Foreclosure Layout.csv','ForeclosureLayout',headers,0)

#######Recorder
#Getting each column's information from REALTYTRAC DLP 3.0 Recorder Layout.xlsx
eachColumn = columnInformation('REALTYTRAC DLP 3.0 Recorder Layout.csv',1)
headers = eachColumn[1]#headers
ignoreMe = makingTxtfile('REALTYTRAC DLP 3.0 Recorder Layout.csv','RecorderLayout',headers,1)
```

```
434
435	#######TaxAssessor
436	#Getting each column's information from REALTYTRAC DLP 3.0 Assessor NO Geo Layout.xlsx
437	eachColumn = columnInformation('REALTYTRAC DLP 3.0 Assessor NO Geo Layout.csv',2)
438	headers = eachColumn[1]#headers
439	ignoreMe = makingTxtfile('REALTYTRAC DLP 3.0 Assessor NO Geo Layout.csv','TaxAssessorLayout',headers,2)
440
441	#################    Checking 'unique Id' that represents each row
442	#Only for Foreclosure data
443	with open('UmichForeclosure_Total.csv','r') as fc1:
444		fc1=fc1.readlines()[1:]
445	dd={}
446	for i in fc1:
447		#9th element is 'unique id'
448		if i.split(',')[8] not in dd:
449			dd[i.split(',')[8]]=1
450		else:
451			dd[i.split(',')[8]]+=1
452	print len(fc1)		#429587
453	print len(dd.keys())	#429587
454
455
456
457
458
459	#############################################################################################################
460	##########################    Step 3    ###################################################################
461	#############################################################################################################
462
463	###############   Making databases to look up index(row) numbers
464
```

```
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
#0 saPropetyID , 7 srUniqueID
UMF=makingListForACertainColumn_Integer('UmichForeclosure_Total.csv',[0,7])
print '1 / 3 completion'
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
#0 saPropetyID , 171 srUniqueID
UMTA=makingListForACertainColumn_Integer('UMichTaxAssessor_Total.csv',[0,171])
print '2 / 3 completion'
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
#0 srUniqueID, 1 saPropetyID
UMR=makingListForACertainColumn_Integer('UMichRecorder_Total.csv',[0,1])
print '3 / 3 completion'
print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")

with sqlite.connect(r'db_saPropertyID_srUniqueID.db') as con:
        cur = con.cursor()
        cur.execute("DROP TABLE IF EXISTS f")
        cur.execute("CREATE TABLE f (row INTEGER PRIMARY KEY,  saPropertyID int, srUniqueID int)")
        cur.executemany("INSERT INTO f (saPropertyID,srUniqueID) VALUES (?,?)", UMF)
        con.commit()
        cur.execute("DROP TABLE IF EXISTS ta")
        cur.execute("CREATE TABLE ta (row INTEGER PRIMARY KEY,  saPropertyID int, srUniqueID int)")
        cur.executemany("INSERT INTO ta (saPropertyID, srUniqueID) VALUES (?,?)", UMTA)
        con.commit()
        cur.execute("DROP TABLE IF EXISTS r")
        cur.execute("CREATE TABLE r (row INTEGER PRIMARY KEY,  srUniqueID int, saPropertyID int)")
        cur.executemany("INSERT INTO r (srUniqueID,saPropertyID) VALUES (?,?)", UMR)
        con.commit()

################    Database exploratory

```

```
with sqlite.connect(r'db_saPropertyID_srUniqueID.db') as con:
        cur = con.cursor()
        To see how many of them do not contain 0
        check01=cur.execute("SELECT f.row,f.srUniqueID FROM f WHERE f.srUniqueID!=0")
        print len(check01.fetchall())#0
        check02=cur.execute("SELECT r.row,r.saPropertyID FROM r WHERE r.saPropertyID!=0")
        print len(check02.fetchall())#7056997
        #one way checking f.row r.row with saPropertyID

        #Double counting (because properties have been reported more than once)
        check1=cur.execute("SELECT f.row,r.row FROM f JOIN r ON (f.saPropertyID = r.saPropertyID ) WHERE r.saPropertyID!=0
ORDER BY f.row,r.row")
        print len(check1.fetchall())#2,645,169

        #two way checking with ta table
        check2=cur.execute("SELECT f.row,r.row FROM ta JOIN r JOIN f ON (f.saPropertyID = ta.saPropertyID and ta.srUniqueID =
r.srUniqueID) ORDER BY f.row,r.row")
        print len(check2.fetchall())#384,446

        #Below is unique case for each property where we used GROUP BY METHOD

        #one way checking
        selection1=cur.execute("SELECT f.row, r.row FROM f JOIN r ON (f.saPropertyID = r.saPropertyID ) WHERE r.saPropertyID!=0
GROUP BY f.saPropertyID ORDER BY f.row,r.row")
        print len(check1.fetchall())#251,755

        #two way checking using fc table with saPropertyID
        selection21=cur.execute("SELECT f.row, r.row, ta.row FROM f JOIN r JOIN ta ON (f.saPropertyID = r.saPropertyID and
f.saPropertyID = ta.saPropertyID ) WHERE r.saPropertyID!=0 GROUP BY f.saPropertyID ORDER BY f.row,r.row,ta.row")
        print len(selection21.fetchall())#251,755

```

```
        # two way checking using r table
        selection22=cur.execute("SELECT f.row, r.row, ta.row  FROM f JOIN r JOIN ta ON (f.saPropertyID = r.saPropertyID and
r.srUniqueID = ta.srUniqueID ) WHERE r.saPropertyID!=0 GROUP BY f.saPropertyID ORDER BY f.row,r.row,ta.row ")
        print len(selection21.fetchall())#228,464

        #two way checking using ta table
        selection23=cur.execute("SELECT f.row,r.row,  ta.row  FROM f JOIN r JOIN ta ON (f.saPropertyID = ta.saPropertyID and
ta.srUniqueID = r.srUniqueID) WHERE ta.saPropertyID!=0 and ta.srUniqueID!=0 GROUP BY f.saPropertyID ORDER BY
f.row,r.row,ta.row ")
        print len(selection22.fetchall())#228,461




################  Making 3 csv files for forclosure and recorder using selection21   ################

#Using selection21
with sqlite.connect(r'db_saPropertyID_srUniqueID.db') as con:
        cur = con.cursor()
        selection21=cur.execute("SELECT f.row, r.row, ta.row FROM f JOIN r JOIN ta ON (f.saPropertyID = r.saPropertyID and
f.saPropertyID = ta.saPropertyID ) WHERE r.saPropertyID!=0 GROUP BY f.saPropertyID ORDER BY f.row,r.row,ta.row")
        selection21Inds = zip(*selection21.fetchall())

with open('JanFeb/UMichForeclosure_Total.csv','r') as fc:
        fc=fc.readlines()
        with open('JanFeb/selection21Foreclosure.csv','w') as wo:
                wo.write(fc[0])
                for i in map(lambda j: fc[j], selection21Inds[0]):
                        wo.write(i)
with open('JanFeb/UMichRecorder_Total.csv','r') as fc:
        fc=fc.readlines()
```

```
        with open('JanFeb/selection21Recorder.csv','w') as wo:
                wo.write(fc[0])
                for i in map(lambda j: fc[j], selection21Inds[1]):
                        wo.write(i)
with open('JanFeb/UMichTaxAssessor_Total.csv','r') as fc:
        fc=fc.readlines()
        with open('JanFeb/selection21TaxAssessor.csv','w') as wo:
                wo.write(fc[0])
                for i in map(lambda j: fc[j], selection21Inds[2]):
                        wo.write(i)
print 'Done!'



#########################################################################################################
#########################    Step 4    ##################################################################
#########################################################################################################

###############   Creating CSV file for foreclosure properties address
with open('JanFeb/selection21Foreclosure.csv','r') as fc1:
        fc1=fc1.readlines()[1:]
with open('JanFeb/selection21Recorder.csv','r') as fc2:
        fc2=fc2.readlines()[1:]
with open('JanFeb/selection21TaxAssessor.csv','r') as fc3:
        fc3=fc3.readlines()[1:]

print len(fc1),len(fc2),len(fc3)

with open('JanFeb/selection21ForeclosureAddress.csv','w') as wo:
```

```
	wo.write('SA_PROPERTY_ID,SR_SITE_ADDR_RAW,FT_SITE_CITY,FT_SITE_STATE,FT_SITE_ZIP,SA_VAL_ASS,SA_VAL_MARKET\n
')
	for i in range(len(fc1)):
		k=fc1[i].split(',')
		a=[k[0]]
		b=[fc2[i].split(',')[9]]
		c=map(lambda j: k[j], [55,56,57])
		d=[fc3[i].split(',')[85]]
		e=[fc3[i].split(',')[95]]
		wo.write(",".join(a+b+c+d+e)+'\n')
print 'Done!'




#################Removing empty addresses
#about 15000 were removed, 237541 left
with open('JanFeb/selection21ForeclosureAddress.csv','r') as kw:
	kw=kw.readlines()
listOfAddresses=[i[0] for i in [map(lambda j: i.split(',')[j] if len(i.split(',')[j])!=0 else '', [1]) for i in kw]]
listOfEmptyAddressIndex=[i for i in range(len(listOfAddresses)) if len(listOfAddresses[i])==0]
for i in listOfEmptyAddressIndex[::-1]: #reverse the order so that we can safely delete all elements (order matters)
	del kw[i]
with open('JanFeb/selection21ForeclosureAddress.csv','w') as wo:
	for i in kw:
		wo.write(i)



Then I geocoded all addresses and get the files in zip_coordinates file

```

```
619
620
621 ################    Creating CSV contains zip,tract,year
622
623 ################This is to make sub csv files due to large amount
624 #total of 237541 properties
625 sfc=pd.read_csv('JanFeb/selection21Geocoded.txt')
626 thisChunk=len(sfc.index)/5
627 strt=len(sfc.index)/5
628 init=0
629 leftOver=len(sfc.index)%5
630 for i in range(5):
631         if i==4:
632                 sfc[init:thisChunk+leftOver+1].to_csv('JanFeb/subfiles/sfc'+str(i)+'.txt', encoding='utf-8',index=False)
633         else:
634                 sfc[init:thisChunk].to_csv('JanFeb/subfiles/sfc'+str(i)+'.txt', encoding='utf-8',index=False)
635                 init=thisChunk
636                 thisChunk+=strt
637
638 ################Getting Census Tract using API
639 #Id-row-number
640 with open('JanFeb/selection21Foreclosure.csv','r') as fc1:
641         fc1=fc1.readlines()
642 dict_id_row={}
643 rowNumber=1
644 for i in fc1[1:]:
645         dict_id_row[i.split(',')[0]]=[rowNumber]
646         rowNumber+=1
647 print os.listdir('JanFeb/subfiles')[5:6]
648 for filename in os.listdir('JanFeb/subfiles')[5:6]:
649         print filename
```

```
650             print datetime.datetime.now().strftime("%Y-%m-%d %H:%M")
651             dict_id_coordinates={}
652             dict_id_propertyValues={}
653             with open('JanFeb/subfiles/'+filename,'r') as ck:
654                     ck=ck.readlines()
655             for i in ck[1:]:
656                     z=i.replace('\r\n','').replace('\n','').split(',')
657                     dict_id_coordinates[z[3]]=(z[2],z[1])
658                     dict_id_propertyValues[z[3]]=(z[4],z[5])
659             base='http://data.fcc.gov/api/block/2010/find'
660             ak= open('Zip_Year_Tract_after8568'+filename,'w')
661             ak.write('Tract,Zip,Year,saPropertyId,Val_ass,Val_market\n')
662             c=1
663             #for i in sorted(dict_id_coordinates.keys()):
664             for i in dict_id_coordinates:
665                     if c>8568:
666                             zipp=fc1[dict_id_row[i][0]].split(',')[57]#zip
667                             yearr=fc1[dict_id_row[i][0]].split(',')[11][:4]#year
668                             #Getting tract information for each housing unit
669                             option={'latitude': float(dict_id_coordinates[i][0]),'longitude':float(dict_id_coordinates[i][1])}
670                             tractt=re.search(r'<Block FIPS="([0-9]{11})',requests.get(url=base, params=option).text).group(1)
671                             val1=dict_id_propertyValues[i][0]
672                             val2=dict_id_propertyValues[i][1]
673                             ak.write("{},{},{},{},{},{}\n".format(tractt,zipp,yearr,i,val1,val2))
674                             print c,len(ck)
675                     c+=1
676             ak.close()
677     ################ Merging subfiles into one
678     print os.listdir('zip_year_tract')
679     listOfFiles=os.listdir('zip_year_tract')
680     with open("Mar/Zip_Year_Tract_Id.txt",'w') as dk:
```

```
        for i in range(len(listOfFiles)):
                if i!='.DS_Store':
                        if i==1:
                                kk = open("zip_year_tract/"+listOfFiles[i],'r')
                                for j in kk.readlines():
                                        dk.write(j)
                                kk.close()
                        else:
                                kk = open("zip_year_tract/"+listOfFiles[i],'r')
                                for j in kk.readlines()[1:]:
                                        dk.write(j)
                                kk.close()

################    Creating CSV contains County_Zip_TractLevel_totalNumbers
with open('JanFeb/selection21Foreclosure.csv','r') as fc1:
        dict_countyCode_countyName={}
        for i in fc1.readlines()[1:]:
                if i.split(',')[5] not in dict_countyCode_countyName:
                        dict_countyCode_countyName[i.split(',')[5]]=i.split(',')[3]
print '1/3'
with open("Mar/Zip_Year_Tract_Id.txt",'r') as sk:
        sk=sk.readlines()
dict_countyCode_tract={}
dict_tract_totalNumbs={}
dict_tract_dict_year_count={}
dict_tract_zip={}
dict_zip_tract={}
howManydifferentYear=[]
for i in sk[1:]:
        countyCo=i.replace('\n','').split(',')[0][2:5]
        thisOne=i.replace('\n','').split(',')[0]
```

```
712             zipp=i.replace('\n','').split(',')[1]
713             years=i.replace('\n','').split(',')[2]
714             if zipp not in dict_zip_tract:
715                     dict_zip_tract[zipp]=[thisOne]
716             else:
717                     if thisOne not in dict_zip_tract[zipp]:
718                             dict_zip_tract[zipp].append(thisOne)
719
720             if countyCo not in dict_countyCode_tract:
721                     dict_countyCode_tract[countyCo]=[thisOne]
722             else:
723                     if thisOne not in dict_countyCode_tract[countyCo]:
724                             dict_countyCode_tract[countyCo].append(thisOne)
725             if thisOne not in dict_tract_totalNumbs:
726                     dict_tract_totalNumbs[thisOne]=1
727             else:
728                     dict_tract_totalNumbs[thisOne]+=1
729             if thisOne not in dict_tract_dict_year_count:
730                     dict_tract_dict_year_count[thisOne]={}
731                     dict_tract_dict_year_count[thisOne][years]=1
732             else:
733                     if years not in dict_tract_dict_year_count[thisOne]:
734                             dict_tract_dict_year_count[thisOne][years]=1
735                     else:
736                             dict_tract_dict_year_count[thisOne][years]+=1
737             if years not in howManydifferentYear:
738                     howManydifferentYear.append(years)
739             if thisOne not in dict_tract_zip:
740                     dict_tract_zip[thisOne]=zipp
741     print '2/3'
742     with open('Mar/County_Zip_TractLevel_totalNumbers.txt', 'w') as writeIt:
```

```
        writeIt.write('County,Zipcode,Id2,Total,F2004,F2005,F2006,F2007,F2008,F2009,F2010,F2011,F2012,F2013,F2014,F2015\n')
        for i in dict_countyCode_countyName:
                countyName=dict_countyCode_countyName[i]
                listOfTractCode=sorted(dict_countyCode_tract[i])
                totalnumbs=0
                eachyeartotal={}
                for eachTract in listOfTractCode:
                        zipp=dict_tract_zip[eachTract]
                        eachTotal=dict_tract_totalNumbs[eachTract]
                        totalnumbs+=eachTotal
                        writeIt.write('{},{},{},{},'.format(countyName,zipp,eachTract,eachTotal))
                        for eachYear in sorted(howManydifferentYear):
                                if eachYear=='2015':
                                        try:
                                                writeIt.write('{}\n'.format(dict_tract_dict_year_count[eachTract][eachYear]))
                                        except:
                                                writeIt.write('0\n')
                                else:
                                        try:
                                                writeIt.write('{},'.format(dict_tract_dict_year_count[eachTract][eachYear]))
                                        except:
                                                writeIt.write('0,')

print '3/3'




################################################################################################
#########################    Step 5    ########################################################
```

```
774 ##########################################################################################################################
775
776 #Creating FC rates
777
778 ################    Adding Mortgage numbers to County_Zip_TractLevel_totalNumbers.txt
779 lookUp=[i for i in os.listdir('Mar') if 'totalNumbers.txt' in i][0]
780 lookUp=pd.read_csv('Mar/'+lookUp)
781 listOfeachFile=os.listdir('Mar/ACS')
782 # lookUp = lookUp.loc[np.repeat(lookUp.index.values,6)]
783 # createNewDf=lookUp[['Id2','County']]
784 # yearss=[2010,2011,2012,2013,2014,2015]
785 # createNewDf['Year']=yearss * (createNewDf.shape[0]/len(yearss))
786 originalIndexNumber=len(lookUp.index)
787 #'Id2','County','Year','Foreclosure','RealEstateOwned','Total'
788 fYears=['F2010','F2011','F2012','F2013','F2014','F2015']
789 onlyForOnce=0 #this is to make initinal dataframe
790 newColumn1='Real_Estate_Owned'
791 newColumn2='Total'
792 for i in range(len(listOfeachFile)):
793         #this is order of [2010,2011,2012,2013,2014,2015]
794         print listOfeachFile[i]
795         eachDf=pd.read_csv('Mar/ACS/'+listOfeachFile[i],header=None)
796         eachDf.drop(eachDf.index[[0]],inplace=True)
797         eachDf.rename(columns=eachDf.iloc[0],inplace=True)
798         eachDf['Id2']=eachDf['Id2']
799         thislookup=lookUp[['Id2','County']]
800         thislookup['Year']=fYears[i][1:]
801         thislookup['Foreclosure']=lookUp[fYears[i]]
802         if onlyForOnce==0:
803                 onlyForOnce+=1
804                 #adding 'RealEstateOwned'
```

```
                br=False
                while br!=True:
                        for col in range(len(eachDf.columns)):
                                if 'Estimate; MORTGAGE STATUS' in eachDf.iloc[0,col] and 'Housing units with a mortgage' in
eachDf.iloc[0,col]:
                                        thisColumn=eachDf.iloc[0,col]
                                        br=True
                                        # print eachDf.loc[eachDf['Id2']=='26163599000',thisColumn]
                eachDf[newColumn1]=eachDf[thisColumn]
                eachDf['Id2'] = pd.to_numeric(eachDf['Id2'], errors='coerce')
                thislookup1=pd.merge(thislookup, eachDf[['Id2',newColumn1]], left_on='Id2', right_on='Id2',  how='outer')
                thislookup1=thislookup1[:originalIndexNumber]
                #adding 'Total'
                thisColumn=eachDf.iloc[0,3]
                eachDf[newColumn2]=eachDf[thisColumn]
                thislookup1=pd.merge(thislookup1, eachDf[['Id2',newColumn2]], left_on='Id2', right_on='Id2',  how='outer')
                thislookup1=thislookup1[:originalIndexNumber]

                # I will make chunk every loop and rbind the chunk after second loop
                #Id2, County year Foreclosures RealEstateOwned Total
        else:
                br=False
                while br!=True:
                        for col in range(len(eachDf.columns)):
                                if 'Estimate; MORTGAGE STATUS' in eachDf.iloc[0,col] and 'Housing units with a mortgage' in
eachDf.iloc[0,col]:
                                        thisColumn=eachDf.iloc[0,col]
                                        br=True
                                        # print eachDf.loc[eachDf['Id2']=='26163599000',thisColumn]
                eachDf[newColumn1]=eachDf[thisColumn]
                eachDf['Id2'] = pd.to_numeric(eachDf['Id2'], errors='coerce')
```

```
                thislookup11=pd.merge(thislookup, eachDf[['Id2',newColumn1]], left_on='Id2', right_on='Id2',  how='outer')
                thislookup11=thislookup11[:originalIndexNumber]

                thisColumn=eachDf.iloc[0,3]
                eachDf[newColumn2]=eachDf[thisColumn]
                thislookup11=pd.merge(thislookup11, eachDf[['Id2',newColumn2]], left_on='Id2', right_on='Id2',  how='outer')
                thislookup11=thislookup11[:originalIndexNumber]
                thislookup1 = thislookup1.append(thislookup11, ignore_index=True)

thislookup1['Id2']=thislookup1['Id2'].astype(int)
thislookup1['Foreclosure']=thislookup1['Foreclosure'].astype(int)
thislookup1['Real_Estate_Owned_Rate']=thislookup1['Foreclosure']/thislookup1['Real_Estate_Owned'].astype(float)
thislookup1['Overall_Rate']=thislookup1['Foreclosure']/thislookup1['Total'].astype(float)
thislookup1=thislookup1.rename(columns =
{'Id2':'GEOID10','Count':'COUNTY','Year':'YEAR','Foreclosure':'FORECLOSURE','Real_Estate_Owned':'REAL_ESTATE_OWNED','Total':'A
LL_HOUSING','Real_Estate_Owned_Rate':'REAL_ESTATE_OWNED_RATE','Overall_Rate':'ALL_HOUSING_RATE'})
thislookup1.to_csv('Mar/NE_ForeclosureRates.csv', encoding='utf-8',index=False)




#############################################################################################
#########################    Step 6    ####################################################
#############################################################################################

#Creating all property values around 10 counties

ta=pd.read_csv('JanFeb/UMichTaxAssessor_Total.csv')
#SA_PROPERTY_ID, SR_UNIQUE_ID,SA_SITE_CITY,SA_SITE_STATE,SA_SITE_ZIP,TAXYEAR,SA_VAL_ASSD, SA_VAL_MARKET
```

```
866 ta =
867 ta[['SR_UNIQUE_ID','SA_PROPERTY_ID','SA_SITE_CITY','SA_SITE_STATE','SA_SITE_ZIP','TAXYEAR','SA_VAL_ASSD','SA_VAL_MARKET']]
868 ta = ta.drop_duplicates(subset='SA_PROPERTY_ID', keep='first')
869
870 rr=pd.read_csv('JanFeb/UmichRecorder_Total.csv')
871 rr = rr[['SR_UNIQUE_ID','SR_PROPERTY_ID','SR_SITE_ADDR_RAW']]
872 rr = rr.drop_duplicates(subset='SR_PROPERTY_ID', keep='first')
873 print len(rr.index)
874 merged1 = pd.merge(ta, rr , left_on='SA_PROPERTY_ID', right_on='SR_PROPERTY_ID', how="left")
875 #PROPERTY_ID's are all unique
876
877 merged1 = merged1[merged1.SR_SITE_ADDR_RAW.notnull()]
878 merged1 = merged1[merged1.SR_SITE_ADDR_RAW != 0]
879 merged1 = merged1[merged1.SA_VAL_ASSD.notnull()]
880 merged1 = merged1[merged1.SA_VAL_ASSD!=0]
881 print len(merged1.index)
882 print list(merged1)
883 merged1.to_csv('Mar/property_Complete.txt', encoding='utf-8',index=False)
884 #total of about 1,600,000 properties have been geocoded
885
886
887 ###############This is to make subfiles due to too large file
888 #spliting them into 10 subfiles
889 sfc=pd.read_csv('Mar/propertyGeocoded.txt')
890 thisChunk=len(sfc.index)/10
891 strt=len(sfc.index)/10
892 init=0
893 leftOver=len(sfc.index)%10
894 for i in range(10):
895         if i==9:
896                 sfc[init:thisChunk+leftOver+1].to_csv('Mar/subfiles/propertyGeocoded'+str(i)+'.txt', encoding='utf-8',index=False)
```

```
            else:
                sfc[init:thisChunk].to_csv('Mar/subfiles/propertyGeocoded'+str(i)+'.txt', encoding='utf-8',index=False)
                init=thisChunk
                thisChunk+=strt
#Using API to get each Census tract for each coordinates
mergingCensusTractFiles(1,'homeValues.txt')
#Using API to calculate 2015 US dollar amount
usingInflationAPI('homeValues.txt','Mar/NE_PropertyValues.txt')
#Getting mean values for each year for each census tract
dfp=pd.read_csv('Mar/NE_PropertyValues.txt')
del dfp['PROPERTY_ID']
del dfp['ZIP']
dfp=dfp.groupby(['GEOID10','YEAR']).mean().reset_index()
dfp.to_csv('Mar/NE_PropertyValues_mean.txt',index=False)

################## Merging NE datasets
#final version of data
dfp=pd.read_csv('Mar/NE_PropertyValues_mean.txt')
dfp2=pd.read_csv('Mar/NE_ForeclosureRates.csv')
dfp['key1']=1
dfp['key2']=1

dff=pd.merge(dfp2, dfp, on = ['GEOID10','YEAR'],  how='outer')
dff=dff[dff.key1==dff.key2]
del dff['key1']
del dff['key2']
dff = dff.dropna(subset=['COUNTY'])
dff.REAL_ESTATE_OWNED_RATE = dff.REAL_ESTATE_OWNED_RATE.replace(np.nan,'.').replace(np.inf,'.')
dff.ALL_HOUSING_RATE = dff.ALL_HOUSING_RATE.replace(np.nan,'.').replace(np.inf,'.')
dffUntil2014 = dff.loc[dff.YEAR!=2015]
dff2015 = dff.loc[dff.YEAR==2015]
```

```
928 dff2015.VAL_ASS_15 = dff2015.VAL_ASS
929 dff2015.VAL_MARKET_15 = dff2015.VAL_MARKET
930 dff = pd.concat([dffUntil2014, dff2015])
931 dff.to_csv('Apr/NE_HOUSING_FORECLOSURE.csv',encoding='utf-8',index=False)
932
933
934
935 ############################
936 ########macthing all the sales transactions property ID to Census tract
937
938 ################ Merging NE datasets
939
940
941 #Merging two files
942 rr=pd.read_csv('JanFeb/UmichRecorder_Total.csv')
943 rr = rr[['SR_UNIQUE_ID','SR_PROPERTY_ID','SR_DATE_TRANSFER','SR_DATE_FILING','MM_FIPS_COUNTY_NAME']]
944 # #changing yy/mm/dd to yy values
945 rr['SR_DATE_TRANSFER'] = rr['SR_DATE_TRANSFER'].map(lambda x: int(str(x)[:4]))
946 rr['SR_DATE_FILING'] = rr['SR_DATE_FILING'].map(lambda x: int(str(x)[:4]))
947 rr['que'] = np.where((rr['SR_DATE_TRANSFER'] == rr['SR_DATE_FILING']), 1, np.nan)
948
949 kk = pd.read_csv('Mar/NE_PropertyValues.txt')
950 kk = kk[['GEOID10','PROPERTY_ID','ZIP']]
951 kk = kk.drop_duplicates(subset='PROPERTY_ID', keep='first')#we assessed different years so we have some duplicates
952 rr['key1']=1
953 kk['key2']=1
954
955 merged1 = pd.merge(rr, kk , left_on='SR_PROPERTY_ID', right_on='PROPERTY_ID', how="left")
956
957 print 'rr length'
958 print len(rr.index)
```

```
print len(merged1[merged1.key1==merged1.key2].index)
print len(merged1[~(merged1.key1==merged1.key2)].index)
zzz = merged1[merged1.key1==merged1.key2]
del zzz['key1']
del zzz['key2']
zzz[['que','GEOID10','PROPERTY_ID','ZIP']] = zzz[['que','GEOID10','PROPERTY_ID','ZIP']].apply(np.int64)
zzz.to_csv('June/zzz.txt', encoding='utf-8',index=False)

#PROPERTY_ID's are all unique
#merged1  --> columns would be : GEOID , 'SR_DATE_TRANSFER', 'SR_DATE_FILING'
#after merging


rr = pd.read_csv('June/zzz.txt')
print len(rr.index)#6748468
print len(rr[rr.que!=1].index)#2745510 'SR_DATE_TRANSFER', 'SR_DATE_FILING' different years
print len(rr[rr.que==1].index)#4002958 'SR_DATE_TRANSFER', 'SR_DATE_FILING' within a same year
print rr.groupby('SR_PROPERTY_ID')['SR_DATE_TRANSFER'].nunique()
print rr.groupby(['GEOID10','SR_DATE_TRANSFER']).count()


#Step 1. create GeoID10 and county data for a later use
rr = pd.read_csv('June/zzz.txt')
rr = rr[['GEOID10','MM_FIPS_COUNTY_NAME']]
rr = rr.drop_duplicates(subset='GEOID10', keep='first')
rr.to_csv('June/CensusTract_County.txt',encoding='utf-8',index=False)

#Step 2. create two different data : GeoID10, SR_DATE_TRANSFER and GeoID10, SR_DATE_FILING
rr = pd.read_csv('June/zzz.txt')
zzz = rr[['GEOID10','SR_DATE_TRANSFER']]
zzz['COUNT']=1
```

```
zzz= zzz.groupby(['GEOID10','SR_DATE_TRANSFER']).count().reset_index()
zzz.to_csv('June/CensusTract_Transfer.txt', encoding='utf-8',heading=True, index=False)

zzz = rr[['GEOID10','SR_DATE_FILING']]
zzz['COUNT']=1
zzz= zzz.groupby(['GEOID10','SR_DATE_FILING']).count().reset_index()
zzz.to_csv('June/CensusTract_Filing.txt', encoding='utf-8',heading=True, index=False)



#Step 3. Calculate 5-yr estimates we are using SR_DATE_FILING (variable contains date it was signed)

listOfACSFile=os.listdir('Mar/ACS')
dataFrameList=[]
for i in listOfACSFile:
        eachDf=pd.read_csv('Mar/ACS/'+i,header=None)
        eachDf.drop(eachDf.index[[0]],inplace=True)
        eachDf.rename(columns=eachDf.iloc[0],inplace=True)
        eachDf['Id2']=eachDf['Id2'].astype('str')
        dataFrameList.append(eachDf[['Id2','Estimate; HOUSING OCCUPANCY - Total housing units']])

rr = pd.read_csv('June/CensusTract_Filing.txt')
rr = rr[rr.SR_DATE_FILING.isin([2006,2007,2008,2009,2010,2011,2012,2013,2014,2015])]
ct_county = pd.read_csv('June/CensusTract_County.txt')
uniqueCensusTract=rr.drop_duplicates(subset='GEOID10',keep='first')
finalData = pd.DataFrame(columns=('GEOID10','COUNTY','YEAR',
'SALES_5_YR_ESTIMATES','ALL_HOUSING','RATES_5_YR_SALES_ESTIMATES'))
rows=0
yearList=[2010,2011,2012,2013,2014,2015]
CTNotMatched=0
for eachCT in list(uniqueCensusTract.GEOID10):
```

```
        eachCTdata=rr.loc[rr['GEOID10'] == eachCT]
        #1567 census tracts
        county = ct_county.loc[ct_county['GEOID10'] == eachCT]['MM_FIPS_COUNTY_NAME'].to_string().split()[1]
        for eachYear in range(len(yearList)):
                #  ## You have to load the ACS 2010, 2011, 2013, ... 2015 data
                kk = eachCTdata.loc[eachCTdata['SR_DATE_FILING'].isin(range(yearList[eachYear]-4,yearList[eachYear]+1))]
                if len(kk.index) == 5:
                        try:
                                totalEstimates = float(dataFrameList[eachYear].loc[dataFrameList[eachYear]['Id2'] ==
str(eachCT)]['Estimate; HOUSING OCCUPANCY - Total housing units'].to_string().split()[1])
                                finalData.loc[rows] =
[eachCT,county,yearList[eachYear],sum(kk.COUNT)/5,totalEstimates,(sum(kk.COUNT)/5)/totalEstimates]#[eachCT,county,str(yearLis
t[eachYear]-4)+'-'+str(yearList[eachYear]),sum(kk.COUNT)/5,totalEstimates,(sum(kk.COUNT)/5)/totalEstimates]
                                rows+=1
                        except:
                                CTNotMatched+=1
print CTNotMatched
finalData.RATES_5_YR_SALES_ESTIMATES = finalData.RATES_5_YR_SALES_ESTIMATES.replace(np.inf,'.')
aa = finalData.loc[finalData.COUNTY!='ST.']
bb = finalData.loc[finalData.COUNTY=='ST.']
bb.COUNTY = 'ST. CLAIR'
finalData = pd.concat([aa, bb])
finalData.to_csv('June/NE_HOUSING_SALES.csv', encoding='utf-8', index=False)



################################################################################################################
#########################    Step 7    ########################################################################
################################################################################################################

foresclosures = pd.read_csv('June/NE_HOUSING_FORECLOSURE.csv')
```

```
sales = pd.read_csv('June/NE_HOUSING_SALES.csv')
foresclosures['GEOID10'] = foresclosures.GEOID10.astype(int)
sales['GEOID10'] = sales.GEOID10.astype(int)
sales['COUNTY'] =
sales['COUNTY'].map({'GENESEE':'Genesee','OAKLAND':'Oakland','LIVINGSTON':'Livingston','LAPEER':'Lapeer','MACOMB':'Macomb','
WAYNE':'Wayne','MONROE':'Monroe','WASHTENAW':'Washtenaw','ST. CLAIR':'Saint Clair'})
yearList=[2010,2011,2012,2013,2014,2015]
dfList=[]
for eachYear in yearList:
        salesSplit = sales[sales['YEAR'] == eachYear]
        fcSplit = foresclosures[foresclosures['YEAR'] == eachYear]
        merged = pd.merge(salesSplit, fcSplit, on=['GEOID10','YEAR','ALL_HOUSING'], how="outer")
        merged.drop_duplicates(subset='GEOID10',inplace=True)
        countyList_x = list(merged.COUNTY_x)
        countyList_y = list(merged.COUNTY_y)
        countyList =[]
        for i in range(len(countyList_x)):
                if type(countyList_y[i]) == str:
                        countyList.append(countyList_y[i])
                else:
                        countyList.append(countyList_x[i])
        del merged['COUNTY_x']
        del merged['COUNTY_y']
        merged['COUNTY'] = countyList
        replacingPdColumn(merged,'SALES_TRANSACTION_ESTIMATES','SALES_5_YR_ESTIMATES')
        replacingPdColumn(merged,'SALES_TRANSACTION_RATES','RATES_5_YR_SALES_ESTIMATES')
        merged =
merged[['GEOID10','COUNTY','YEAR','FORECLOSURE','REAL_ESTATE_OWNED','SALES_TRANSACTION_ESTIMATES','ALL_HOUSING','RE
AL_ESTATE_OWNED_RATE','ALL_HOUSING_RATE','SALES_TRANSACTION_RATES','VAL_ASS','VAL_MARKET','VAL_ASS_15','VAL_MAR
KET_15']]
        dfList.append(merged)
```

```
newDf=pd.concat(dfList)
newDf['GEOID10'] = newDf.GEOID10.astype(int)
newDf['YEAR'] = newDf.YEAR.astype(int)
newDf['ALL_HOUSING'] = newDf.ALL_HOUSING.astype(int)
newDf = newDf.replace(np.nan,'.', regex=True)

newDf.rename(columns={'FORECLOSURE': 'FORECLOSURES', 'REAL_ESTATE_OWNED':
'REAL_ESTATE_OWNED_MORTGAGES','REAL_ESTATE_OWNED_RATE':'FORECLOSURES_BY_REAL_ESTATE_OWNED_MORTGAGES','AL
L_HOUSING_RATE':'FORECLOSURES_BY_ALL_HOUSING','SALES_TRANSACTION_ESTIMATES':'SALES_TRANSACTIONS','SALES_TRANSA
CTION_RATES':'SALES_TRANSACTIONS_BY_ALL_HOUSING'}, inplace=True)

newDf['deno'] = newDf['REAL_ESTATE_OWNED_MORTGAGES'].replace('.', -10,regex=True).astype(float)
newDf['nume'] = newDf['ALL_HOUSING'].replace('.', -100,regex=True).astype(float)


newDf['REAL_ESTATE_OWNED_MORTGAGES_BY_ALL_HOUSING']= newDf['nume']/newDf['deno']
del newDf['deno']
del newDf['nume']

newDf =
newDf[['GEOID10','COUNTY','YEAR','ALL_HOUSING','REAL_ESTATE_OWNED_MORTGAGES','REAL_ESTATE_OWNED_MORTGAGES_BY
_ALL_HOUSING','FORECLOSURES','FORECLOSURES_BY_REAL_ESTATE_OWNED_MORTGAGES','FORECLOSURES_BY_ALL_HOUSING','S
ALES_TRANSACTIONS','SALES_TRANSACTIONS_BY_ALL_HOUSING','VAL_ASS','VAL_MARKET','VAL_ASS_15','VAL_MARKET_15']]

print newDf.head()

newDf.to_csv('June/NE_HOUSING.csv', encoding='utf-8', index=False)
```